Apple IIGS
#42: Custom Windows

Written by: Dan Oliver & Keith Rollin November 1988

This Technical Note describes custom windows which are now supported with Window Manager version 2.2. This Note supersedes all prior documentation on custom windows.

With Window Manager version 2.2 or later, which is available on Apple IIGS System Disk 3.2 and later, you may now define your own type of window or window shape, such as a round or hexagonal window. You also may define a window which performs tasks that would normally be handled by an application.

To define your own type of window, a custom window, you must write a routine that performs some window functions. This routine is a window definition procedure (`defProc`), and in this case it is a custom window `defProc`. When the Window Manager needs to do something window specific, it calls your `defProc`.

The window `defProc` is a good part of the Window Manager, and writing one is not an easy task. A window `defProc` must perform complicated tasks that are very dependent on the state of the machine, and it must be very careful not to disturb the state of the machine. One of the problems in writing a `defProc` is knowing when it can do something and when it cannot. It is almost impossible to document all of the combinations of calls that you can or cannot make from one part or another of the `defProc`, and even if all cases were found, the resulting document would read like something from an obscure government bureau and probably be even harder to understand.

Now that you know writing a `defProc` is tough, here's how to make things as easy as possible. Try to understand how the system interacts with the `defProc` and work with the system. For example, a `defProc` is called to hit test window parts when the user presses the mouse button. The Window Manager will pass that part back to the `defProc` to perform drawing while the Window Manager is tracking the pressed button. The `defProc` could keep control when asked to hit test and perform the tracking itself, but since this is not how the system is designed to work, your `defProc` will be hard to write, may not ever work correctly, and may break in future versions of the Window Manager. Try to stay on the path outlined in this Technical Note. Also understand that the interface to definition procedures is as general as possible to allow them to perform tasks which are as yet unknown. To allow for this future growth, the outlined path is not always a clear path.

Another way to make things easier is to write conservative code. Do not assume things like the data bank being set to something nice when the `defProc` is called or the caller restoring the direct page pointer upon return if you have changed it. Use caution. A `defProc` can be very difficult to debug because it is not very linear and can be called when you least expect.

Interaction Between the Window Manager and TaskMaster

The Window Manager and `TaskMaster` actually do much less than many people think since window definition procedures perform most of the tasks. The definition procedures handle such things as title bars, information bars, and scroll bars, while the Window Manager and `TaskMaster` support these things by passing requests to the `defProc` in standard ways. The Window Manager knows that windows have some shape, overlap, may contain parts, may be invisible, and are created and deleted, but it does not know much else. `TaskMaster` knows to call `GetNextEvent` and performs some tasks, but much of what many people consider `TaskMaster` is contained in the standard document window `defProc`. In addition to the list mentioned above, the `defProc` handles calling `TrackGoAway` and scrolling the content. The remainder of this Note describes what is expected of a `defProc` and when.

Telling the Window Manager About Your Window

You tell the Window Manager about your custom window when `NewWindow` creates it. Instead of passing the parameter list defined in `NewWindow`, you pass a pointer to a custom window parameter list. A custom window parameter list is defined as follows:

`paramID`	WORD	ID of parameter list, zero for custom.
`newDefProc`	LONG	Address of your custom defProc.
`newData`	BYTE[n]	Additional data defined by your defProc.

`NewWindow` checks the `paramID` field and calls your `defProc` with the pointer to the parameter list. See the `wNew` operation under Calling the Custom DefProc for more information.

Once `NewWindow` creates the window, the Window Manager will always know that it is defined by your `defProc`.

Calling the Custom defProc

A window `defProc` is called with the following items on the stack:

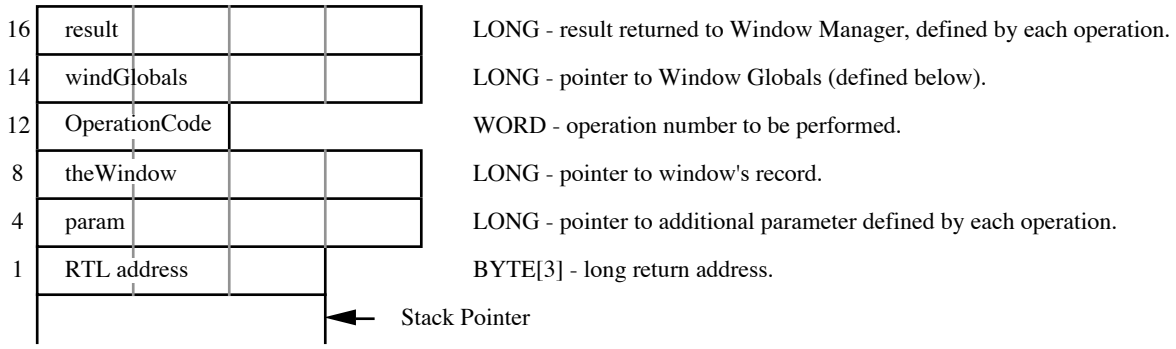


Figure 1 – Stack Prior to Calling a Window defProc

The defProc must return with the carry flag clear if there was no error or with the carry flag set and the y register set with an error code if there was an error.

Window globals (`windGlobals`) is a pointer to a table of variables which the Window Manager maintains for use by the `defProc`. The table is defined as follows:

`lineW`	WORD	Width of vertical lines (size depends on video mode).
`titleHeight`	WORD	Height of a standard title bar.
`titleYPos`	WORD	Y offset for the title (in system font) to center in a standard title bar.
`closeHeight`	WORD	Height of the close box icon.
`closeWidth`	WORD	Width of the close box icon.
`defWindClr`	LONG	Pointer to the default window color table.
`windIconFont`	LONG	Handle of the current window icon font.
`screenMode`	WORD	TRUE if 640 mode, FALSE if 320 mode.
`pattern`	BYTE[32]	Temporary pattern buffer.
`callerDpage`	WORD	Direct page pointer of the last caller to `TaskMaster`.
`callerDataB`	WORD	Data bank of the last caller to `TaskMaster` (bank in both bytes).

Operation numbers are as follows (each operation is described later in its own section):

`wDraw`	0	Draw the window's frame.
`wHit`	1	Tell in what region the mouse button was pressed.
`wCalcRgns`	2	Calculate `wStrucRgn` and `wContRgn`.
`wNew`	3	Complete the creation of a window.
`wDispose`	4	Complete the disposal of a window.
`wGetDrag`	5	Return address that will draw the outline of the window while dragging.
`wGrowFrame`	6	Draw the outline of a window being resized.
`wRecSize`	7	Return size of the additional space needed in the window record.
`wPosition`	8	Return `RECT` that is the window's `portRect`.
`wBehind`	9	Return where the window should be placed in the window list.
`wCallDefProc`	10	Generic call to a `defProc`, defined by the `defProc`.

wDraw, Operation 0

The `wDraw` operation draws the window's frame and is only called for visible windows. This operation draws in local coordinates in the current `GrafPort`, which is the Window Manager's `GrafPort`. When the drawing is finished, the only states of the `GrafPort` that may have changed are the pen pattern, the fill pattern, and the pen size, as all other states must be the same as when the `defProc` was called. This means that if you change the font to print some text, you

must save and restore the original font. For the pen, `PenNormal` will restore the pen to an acceptable state.

`Param` is defined as follows:

Bit 31	1 to highlight the indicated part, 0 to unhighlight.
Bits 0-30	The part to draw (either highlighted or unhighlighted):
	0 Draw the window's entire frame, including any frame controls and the items listed below. Note that you should check the window's `fHilited` flag to determine how to draw the frame.
	1 Draw the go-away region.
	2 Draw the zoom region.
	3 Draw the information bar.

`Result` returned must be zero and the carry flag must be clear.

The Window Manager will draw the content.

Need to Redraw Your Window?

If your custom window `defProc` gets called to change some item in its window record (see `wCallDefProc` below), you may want to redraw your window. For instance, if your application makes a `SetWTitle` call, you would want to draw the name of the new title on the screen.

The routine `wCallDefProc` can call the `wDraw` routine to do this drawing. However, it should bracket the calls to `wDraw` with two Window Manager calls that save and restore some internal variables:

`StartFrameDrawing`	$5A0E
`PUSH:LONG`	Pointer to the window record (not the `GrafPort`)

This call does the setup for drawing a window frame and is only called by a window definition procedure before drawing the frame. You should call `EndFrameDrawing` when finished drawing.

`EndFrameDrawing`	$5B0E
No input or output	

This call restores the Window Manager variables after a call to StartFrameDrawing and is only called by a window definition procedure after drawing a window frame.

wHit, Operation 1

The `wHit` operation is called to hit test the window's frame. Given a set of screen coordinates, this operation should return what part, if any, of the window is at that coordinate. This operation is only called for visible windows. The current port will be that of the Window Manager and the window frame will be in local coordinates.

`Param` is defined as:

Bits 0-15	Vertical (Y) coordinate in local coordinates.
Bits 16-31	Horizontal (X) coordinate in local coordinates.

`Result` returned must be one of the following values and the carry flag must be clear:

`wNoHit`	0	Not on the window at all.
`wInDrag`	20	Coordinates are in the window's drag region (title bar).
`wInGrow`	21	Coordinates are in the window's grow region (size box).
`wInGoAway`	22	Coordinates are in the window's go-away region (close box).
`wInZoom`	23	Coordinates are in the window's zoom region (zoom box).
`wInInfo`	24	Coordinates are in the window's information bar.
`wInFrame`	27	Coordinates are in the window, but not in any of the other areas.

xx Any code the application can handle (bit 15 is reserved for theWindow Manager)

wCalcRgns, Operation 2

The `wCalcRgns` operation, which is called only for visible windows, is used to calculate the window's entire region (frame plus content called `StrucRgn`) and just its content region (called `ContRgn`). Both regions must be set to global coordinates, and both will already be allocated with their handles stored in the window record's `wStrucRgn` and `wContRgn` fields.

Use the `portRect` and the `boundsRect` of the window's `GrafPort` to calculate these two regions. The port will have been set from the information passed to `NewWindow` along with any size changes. A method for obtaining the global `RECT` of the content is given below. Refer to the QuickDraw II chapter in the *Apple IIGS Toolbox Reference* for a full description of ports. When calculating the regions, do not change the clip region (`ClipRgn`) or the visible region (`VisRgn`) of the `GrafPort`.

`Param` is not defined and should not be used.

`Result` returned must be zero and the carry flag must be clear.

```
IN:    window = pointer to window record.
OUT:   rect = global RECT of window's content.

       ldy #wPort+portRect+y1
       lda [<window],y
       ldy #wPort+portInfo+boundsRect+y1
       sec
       sbc [<window],y
       sta <rect+y1
;
       ldy #wPort+portRect+x1
       lda [<window],y
       ldy #wPort+portInfo+boundsRect+x1
       sec
       sbc [<window],y
       sta <rect+x1
;
       ldy #wPort+portRect+y2
       lda [<window],y
       ldy #wPort+portInfo+boundsRect+y1
       sec
       sbc [<window],y
       sta <rect+y2
;
       ldy #wPort+portRect+x2
       lda [<window],y
       ldy #wPort+portInfo+boundsRect+x1
       sec
       sbc [<window],y
       sta <rect+x2
```

Although there are other ways to obtain the global `RECT` of the content, this example gives the correct method. You should never rely on the top and left side of the `portRect` being zero.

wNew, Operation 3

The wNew operation is called to perform any additional initialization that may be required for a custom window. The following items are already done for the window:

- If a window record is supposed to be allocated, it is. All fields, other than those fields listed below, are set to zero
- A port opens in the window record's wPort field.
- The window is added to the Window Manager's window list, and the wNext field is set.
- The wDefProc, wStrucRgn, wContRgn and wUpdate regions are set with the handles of the allocated regions. It is the responsibility of the defProc to define the shape of the wStrucRgn and wContRgn regions.
- The fAllocated and fHilited bits in the wFrame field of the window record are set (see the window record definition for a definition of these bits) and should not be disturbed; all other bits in wFrame are set to zero. The defProc should set the fCtlTie, fVis and fQContent bits, and it can set and use other bits in the wFrame field as it wishes.
- It is the responsibility of the defProc to set the wRefCon, wContDraw, and wFrameCtls fields, the bits already mentioned in the wFrame field, and any other fields which it defines in the wCustom part of the window record.

Param is a pointer to the parameter list pointer which was passed to NewWindow.

Result returned must be zero and the carry flag must be clear.

wDispose, Operation 4

The wDispose operation is called to perform any additional disposal that may be required of a custom window. This operation is called before the Window Manager performs any disposal actions on the window.

Param is not defined and should not be used.

Result should be FALSE to continue disposal or TRUE to abort the disposal. In either case, the carry flag should be clear. Returning TRUE would be very unusual and should be carefully thought out. After returning FALSE, the Window Manager will erase the window, remove the window from the Window Manager's window list, free any controls in the window's wControls and wFrameCtl lists, free the handles in the wStrucRgn, wContRgn and wUpdateRgn fields, close the window's GrafPort, and free its record if it is allocated (see the wFrame field).

wGetDrag, Operation 5

The wGetDrag operation is called to get the address of a routine that will draw an outline of the window.

Param is not defined and should not be used.

Result returned must be the address of a frame outline routine or zero for a default frame; the default frame is the bounds RECT of the strucRgn. The frame outline routine is called from DragRect with dragRectPtr set to the bounds RECT of the strucRgn. Your routine is called with the following parameters:

PUSH:WORD - delta X
PUSH:WORD - delta Y
PUSH:BYTE[3] - return address

Your routine should draw or erase the outline of the object in its new position using the passed deltas. You have several different methods of determining whether to erase or draw and how to compute the position of the object, the easiest method being to draw the outline using XOR mode. The first time your routine is called, you draw. The next time your routine is called, you erase. Your routine should draw in the current port. The current pen pattern will be the pattern pointed to by dragPatternPtr from DragRect and the pen mode is XOR.

You also need to know where to draw the outline. One way is to offset the starting RECT (dragRectPtr) by the given deltas. You should make a copy of the bounds RECT of the strucRgn when wGetDrag is called. Modify that rectangle with the deltas to obtain the rectangle to frame.

wGrowFrame, Operation 6

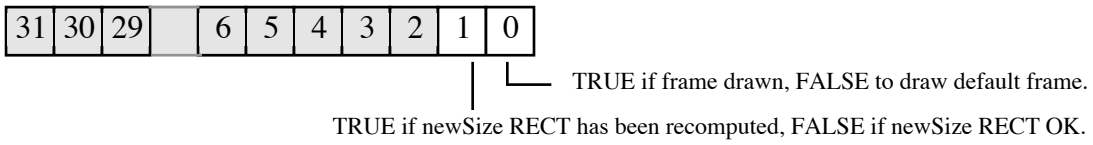
The wGrowFrame operation is called to draw an outline of the window when the window is being resized.

This operation should use the current port, pen pattern, and pen mode. The frame should be drawn with only the following QuickDraw II calls: Line, LineTo, FrameRect, FrameRgn, FramePoly, FrameOval, FrameRRect, and FrameArc (the Invert equivalents to Frame could also be used). You want to use the current GrafPort setting with only certain QuickDraw II calls since this routine will be called an even number of times; the first time it is called to draw the frame and the next time to erase that which it drew the first time. If it needs to use QuickDraw II calls other than those listed above, this operation handler could keep track of odd and even calls to know whether to draw or erase the frame.

Param is a pointer to the following parameter list:

newSize	RECT	Rectangle that defines the new size.
drawFlag	WORD	TRUE to draw the frame, FALSE to erase.
startRect	RECT	Bounds of wStrucRgn when dragging started.
deltaY	WORD	Vertical movement since starting to drag (signed).
deltaX	WORD	Horizontal movement since starting to drag (signed).

Result should be:



The Window Manager assumes that the frame of the grow outline is the same as the bounds of the window's `wStrucRgn`. This `RECT` is stored in the `startRect` of the parameter list and does not change through out the dragging. The next assumption is that the window grows from the lower right corner. As the cursor moves, the lower right corner of the `RECT` in `newSize` changes. However, if these assumptions are not correct for a custom window they can be overridden by changing the `RECT` in `newSize` (by using `startRect` or the window's record and the deltas) and returning TRUE for bit 1 in `Result`. The carry flag should return clear.

wRecSize, Operation 7

The `wRecSize` operation is called to ask how large a window record should be allocated.

Note: The window pointer passed in `theWindow` is not valid for this call.

`Param` is the parameter list pointer that is passed to `NewWindow`.

`Result` is the number of additional bytes required in the window record. The standard window record header will always be allocated.

Example:

If your custom window needs a one word field in the window record for your own use you would return 2 in `Result`. The Window Manager takes `Result` and adds to it the size of the standard record header of 212 bytes and allocates a window record that is 214 bytes long in this case. Your one word field is at the end of the standard window record header with an offset of 212 bytes.

If there is some error, return the carry flag set with an error code in the y register, which will cause `NewWindow` to abort and return the error code to the application which called it. If there is no error, return the carry flag clear.

Window Record Already Allocated?

If the window record is already allocated then `Result` should be the pointer to the window record with bit 31 of the pointer set to TRUE. Generally, window records are allocated (refer to Window Record Definition at the end of this Note for more information about window records).

wPosition, Operation 8

`Param` is the parameter list pointer that is passed to `NewWindow`.

`Result` is a pointer to the `RECT` that will be the window's `portRect`, and you should return the carry flag clear.

wBehind, Operation 9

`Param` is the parameter list pointer that is passed to `NewWindow`.

`Result` is where the window should be placed in the window list. A long $FFFFFFFF means insert the window as the top window while a long $00000000 means to insert it as the bottom window. Any other value is a pointer to the window behind which this window should be placed. You should return the carry flag clear.

wCallDefProc, Operation 10

WCallDefProc is a generic call to the defProc that is defined by the defProc. With this call a window defProc can define many special functions.

The input to the defProc is:

param = pointer to the following parameter table:

dRequest	WORD	Requested operation number.
paramID	WORD	Parameter block type: $0000-$7FFF reserved by system ($0000 defined below). $8000-$FFFF reserved for custom defProcs.
newParam	BYTE[n]	New parameter field used by some operations.

The paramID field defines dRequest, which in turn defines newParam and the result of the wCallDefProc call. You can think of dRequest as the operation number passed to the defProc. Here is an example of how the paramID defines dRequest: if paramID is zero, dRequest 3 is defined as wSetPage (defined below); but if paramID is $8345 (or any number other than zero), dRequest 3 could be defined as something entirely different.

The following dRequest values are defined for wCallDefProc operations with a paramID of zero. Your defProc should check for handling only these codes. In the future, codes 34 and greater may be defined, and your defProc should know not to handle them.

wSetOrgMask	0	wGetInfoDraw	17
wSetMaxGrow	1	wGetOrigin	18
wSetScroll	2	wGetDataSize	19
wSetPage	3	wGetZoomRect	20
wSetInfoRefCon	4	wGetTitle	21
wSetInfoDraw	5	wGetColorTable	22
wSetOrigin	6	wGetFrameFlag	23
wSetDataSize	7	wGetInfoRect	24
wSetZoomRect	8	wGetDrawInfo	25
wSetTitle	9	wGetStartInfoDraw	26
wSetColorTable	10	wGetEndInfoDraw	27
wSetFrameFlag	11	wZoomWindow	28
wGetOrgMask	12	wStartDrawing	29
wGetMaxGrow	13	wStartMove	30
wGetScroll	14	wStartGrow	31
wGetPage	15	wNewSize	32
wGetInfoRefCon	16	wTask	33

```
wSetOrgMask          0
      newParam  =  WORD - window's origin mask.
      result    =  None.
```

Called when SetOriginMask is called.

```
wSetMaxGrow          1
```

```
      newParam  =  WORD - maximum window height.
                   WORD - maximum window width.
      result    =  None.
```

Called when `SetMaxGrow` is called.

`wSetScroll` 2

```
      newParam  =  WORD - number of pixels to scroll when arrow is
                   selected.
      result    =  None.
```

Called when `SetScroll` is called.

`wSetPage` 3

```
      newParam  =  WORD - pixels to scroll when page region is selected.
      result    =  None.
```

Called when `SetPage` is called.

`wSetInfoRefCon` 4

```
      newParam  =  LONG - value passed to info bar draw routine
                   (app's use only).
      result    =  None.
```

Called when `SetInfoRefCon` is called.

`wSetInfoDraw` 5

```
      newParam  =  LONG - address of info bar draw routine.
      result    =  None.
```

Called when `SetInfoDraw` is called.

`wSetOrigin` 6

```
      newParam  =  WORD - flag, TRUE to scroll content.
                   WORD - window's Y origin.
                   WORD - window's X origin.
      result    =  None.
```

Called when `SetContentOrigin` is called.

`wSetDataSize` 7

```
      newParam  =  WORD - height of window's data area.
                   WORD - width of window's data area.
      result    =  None.
```

Called when `SetDataSize` is called.

`wSetZoomRect` 8

```
      newParam  =  LONG - pointer to new zoom RECT.
      result    =  None.
```

Called when `SetZoomRect` is called.

`wSetTitle` 9

```
newParam  =  LONG - pointer to new title.
result    =  None.
```

Called when `SetWTitle` is called.

```
wSetColorTable     10
      newParam  =  LONG - pointer to new color table.
      result    =  None.
```

Called when `SetFrameColor` is called.

```
wSetFrameFlag      11
      newParam  =  LONG - pointer to new zoom RECT.
      result    =  None.
```

Called when `SetWFrame` is called.

```
wGetOrgMask        12
      newParam  =  None.
      result    =  WORD - window's origin mask.
```

```
wGetMaxGrow        13
      newParam  =  None.
      result    =  Low word is window's maximum height when grown.
                   High word is window's maximum width when grown.
```

Called when `GetMaxGrow` is called.

```
wGetScroll         14
      newParam  =  None.
      result    =  Low word is number of pixels to scroll when arrow is selected.
```

Called when `GetScroll` is called.

```
wGetPage           15
      newParam  =  None.
      result    =  Low word is pixels to scroll when page region is selected.
```

Called when `GetPage` is called.

```
wGetInfoRefCon     16
      newParam  =  None.
      result    =  Value passed to info bar draw routine.
```

Called when `GetInfoRefCon` is called.

```
wGetInfoDraw       17
      newParam  =  None.
      result    =  Address of info bar draw routine.
```

Called when `GetInfoDraw` is called.

```
wGetOrigin         18
      newParam  =  None.
      result    =  Low word is content's Y origin.
                   High word is content's X origin.
```

Called when `GetContentOrigin` is called.

```
wGetDataSize        19
      newParam  =  None.
      result    =  Low word is window's data height.
                   High word is window's data width.
```

Called when `GetDataSize` is called.

```
wGetZoomRect        20
      newParam  =  None
      result    =  Pointer to window's current zoom RECT.
```

Called when `GetZoomRect` is called.

```
wGetTitle           21
      newParam  =  None
      result    =  Pointer to window's title.
```

Called when `SetWTitle` is called.

```
wGetColorTable             22
      newParam  =  None.
      result    =  Pointer to window's color table.
```

Called when `SetFrameColor` is called.

```
wGetFrameFlag              23
      newParam  =  None.
      result    =  Low word is window's wFrame field.
```

Called when `SetWFrame` is called.

```
wGetInfoRect        24
      newParam  =  LONG - pointer to place to store info bar's enclosing RECT.
      result    =  None.
```

Called when `GetRectInfo` is called.

```
wGetDrawInfo        25
      newParam  =  None.
      result    =  None.
```

Called when `DrawInfoBar` is called.

```
wGetStartInfoDraw 26
      newParam  =  LONG - pointer to place to store info bar's enclosing
                   RECT.
      result    =  None.
```

Called when `StartInfoDrawing` is called.

```
wGetEndInfoDraw            27
      newParam  =  None.
      result    =  None.
```

Called when `EndInfoDrawing` is called.

wZoomWindow 28

```
newParam  =  None.
result    =  None.
```

Called when ZoomWindow is called.

wStartDrawing 29

```
newParam  =  None.
result    =  None.
```

Called when StartDrawing is called.

wStartMove 30

```
newParam  =  WORD - new y position (global).
             WORD - x position (global).
result    =  Low word is new y position (global).
             High word is x position (global).
```

Called before MoveWindow moves a window.

wStartGrow 31

```
newParam  =  None.
result    =  None.
```

Called before GrowWindow tracks the growing of a window.

wNewSize 32

```
newParam  =  LONG - pointer to:
                         WORD - proposed new height.
                         WORD - proposed new width.
                         These two values can be changed.
result    =  Low word TRUE if only uncovered content should be drawn.
             FALSE if entire content should be redrawn.
```

Called by SizeWindow before it resizes a window. The new height and width can be changed by modifying the words pointed to by the pointer in newParam.

wTask 33

```
newParam  =  LONG - pointer to task record.
             WORD - result from FindWindow.
result    =  Low word is code returned by TaskMaster (zero if handled).
             High word is task performed.  Returned in TaskData if code is 0.
```

Called from TaskMaster when it cannot handle a task. If the user presses the mouse button over a window, TaskMaster will call FindWindow to find out what part of the window. TaskMaster will then handle the task if FindWindow returns wInMenuBar or bit 15 of the window pointer is set (system window). Otherwise, the result of FindWindow is passed to wTask to be handled or not.

If the defProc can handle the task it should do so and return zero in the low word of the result (which will be the result to the application returned from TaskMaster) and a code of the task performed in the high word of the result (which is returned to the application in its task record TaskData field). Fields in the task record may also be

modified to return parameters to the application as this is the same record passed to `TaskMaster`.

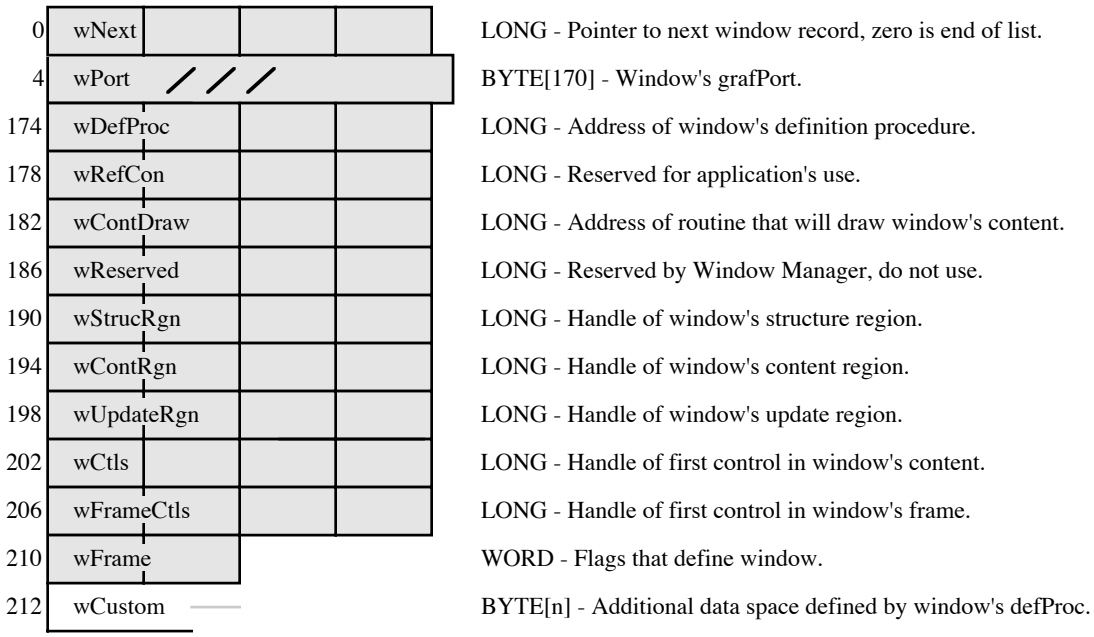
If the `defProc` cannot handle the task, it should return the result from `FindWindow` (the second field in `newParam`) in the low word of the result. The high word of the result is not used.

For example, the standard document window `defProc` handles the following results from `FindWindow` if the `taskMask` record allows.

`wInContent`	Brings the window to the top.
`wInDrag`	Calls `DragWindow`.
`wInGrow`	Brings the window to the top. If it is already on the top, it calls `GrowWindow` and `SizeWindow`.
`wInGoAway`	Calls `TrackGoAway`.
`wInZoom`	Calls `TrackZoom` and `ZoomWindow`.
`wInInfo`	Brings the window to the top.
`wInFrame`	Brings the window to the top. If it is already on the top, checks if it is on one of the window's scroll bars, tracks it, and scrolls the window's content as needed.

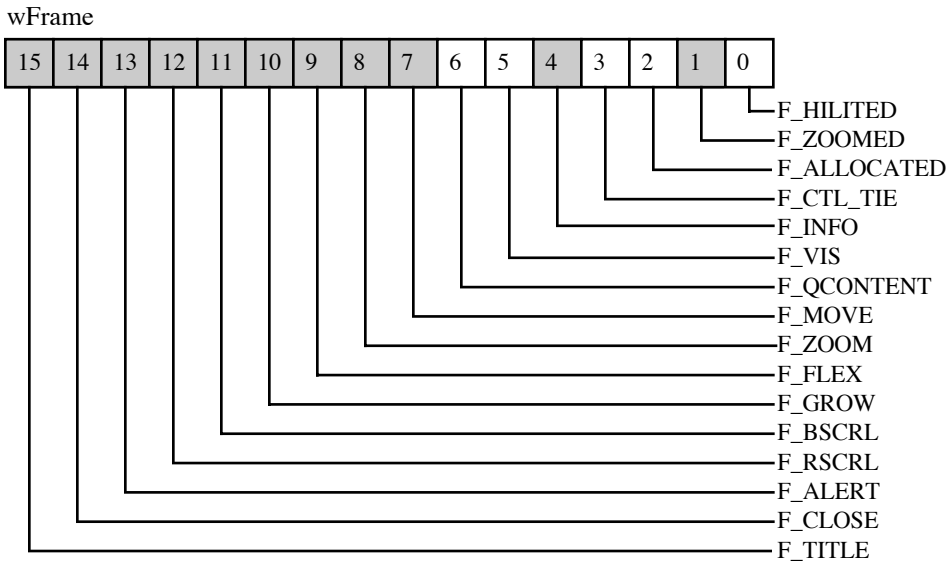
A custom window `defProc` can return any code (bit 15 is used for system windows) it wants when it is called to do a hit test. This code would be that returned by `FindWindow`, and the application would have to know about the code if it called `FindWindow` instead of `TaskMaster`. If `TaskMaster` is used, the code that `FindWindow` returns is passed back to your `defProc` with a `wCallDefProc` and `wTask`. The `defProc` could perform any task it wanted: change colors, eject a disk, run a spelling checker, or anything else.

Window Record Definition



The changes use some vacant space under the window port and add the `wReserved` field to the record for future expansion.

In addition to defining the window record, the `wFrame` field needs to be further defined. In the diagram below the shaded bits are reserved for use by each window `defProc` (the values shown are those used by the standard document window `defProc`). Bits not shaded are reserved by the Window Manager and are applicable to all windows.



Further Reference

- *Apple IIGS Toolbox Reference*, Volume 1
- System Disk 4.0 Release Notes